Implementation of the ASTER science standard data product requirements in the EOSDIS system

B. T. Eng, A. T. Murray, M. Pniel, G. Geller, C. Lee and A. A. Schwartz

Jet Propulsion Laboratory
Pasadena, California 91109 USA

July 1 f), 1996

ABSTRACT

The Advanced Spaceborne Thermal Emission and Reflection Radiometer (ASTER) is designed to provide a high-resolution map of the Earth in both visible, near-infrared, and thermal spectral regions of the electromagnetic spectrum. The ASTER Science Team has developed standard data product algorithms to permit the estimation of surface radiances and reflectance values, to calculate surface temperatures both over water and land, to provide a color enhanced product with a high degree of surface discriminability, in addition to other functions. The ASTER Product Generation System (PGS) Team is implementing these requirements within the constraints of the EOSDIS system, using a rapid development methodology that emphasizes open lines of communication in a team approach using concurrent engineering techniques. The PGS Development environment was structured both to conform to the changing needs of the ECS system and to incorporate experimentation with and modification of the science algorithms as the software was being developed and tested. This challenging environment required a focus on novel methods for requirements tracking, software interface uniformity, Toolkit transparency, and platform independence. This approach required a high degree of interoperability of the software development environment, as well as a flexible and highly integrated configuration management and testing approach. In addition in order to validate the PGS software in the operational environment of the EOSDIS, a remote integration testing approach was adopted to provide a rapid convergence of the final integrated system. This paper describes the critical elements in the development and integration of the ASTER PGS system.

Keywords: ASTER, PGS, EDAAC, Standard Data Products, Remote Testing, Design Methodology

1 INTRODUCTION

The Advanced Spaceborne Thermal Emission and Reflection Radiometer (ASTER)[1] is a multispectral imaging instrument scheduled to fly on the Earth Observing System (EOS) AM-1 Platform. The ASTER instrument includes three telescopes, one for Thermal-Infrared (TIR) wavelengths, one for Short-Wave Infrared (SWIR) and one which covers wavelengths from Visible through Near Infrared (VNIR). Figure 1 shows some of the instrument's technical specifications.

A variety of data products will be produced from ASTER data. This paper will focus on design and methodology issues relating to the higher level Standard Data Products, which are: Surface Radiance (TIR, SWIR and VNIR), Surface Reflectance (VNIR and SWIR), Brightness Temperature at Sensor, Surface Kinetic Temperature, Surface Emissivity and a color-enhanced Decorrelation Stretch product. A high resolution Digital Elevation Model (DEM) product will also be produced from a subset of the acquired ASTER data, using both the nadir-looking and backward-looking VNIR images. The DEM will be produced using a commercial software package and will not be discussed further in this paper.

The ASTER Product Generation System (APGS) is the collective name for the software which will be used to produce all the ASTER higher level data products at a Distributed Active Archive Center (DAAC). The APGS described here uses Level-1 data (radiometrically and geometrically calibrated radiance at the sensor) as its lowest level input. Level-1 processing will be performed in Japan using Japanese software. This paper describes the architecture and implementation of the APGS and presents some of the techniques which are being used to maintain a rapid development cycle while accommodating changes to requirements, algorithms, and software tools. We will also describe our approach to the integration and test process. A section is devoted to describing our experiences during remote integration of the Beta version of the ASTER PGS.

Characteristic	VNIR	SWIR	TIR
Spectral Range	Band 1:0.52-0.60 µm Nadir Looking Band 2:0,63-0.69 µm Nadir Looking Band 3: 0.76 -0.86 µm Nadir Looking Band 3B: 0.76-0.86 µm Backward Looking	Band 4:1.600-1.700 µm Band 5:2.145-2.185 µm Band 6:2.185-2.225 µm Band 7: 2.235 - ? 285 µm Band 8:2 295 - 2.365 µm Band 9:2.360-2.430 µm	Band 10.8.125-8.475 µm Band 11:8.475-8.825 µm Band 1?: 8.925-9.?75 µm Band 13:10.25 - 10.95 µm Band 14: 10. W-11.65 µm
Ground Resolution	15 m	30 m	90 m
Data Rate (Mbits/s)	62	?3	4.?
Cross-1 rack Pointing (deg.)	± 24	± 8.55	± 8.55
Cross-T rack Pointing (km)	± 318	± 116	± 116
Swath Width (km)	60	60	60
Detector Type	Si	PtSi-Si	HgCdTe
Quantization (bits)	8	8	12

Figure 1: ASTER Instrument Characteristics

2 ENVIRONMENT

2.1 ASTER Project

The ASTER project consists of the ASTER Science Team, which is responsible for science algorithm specification and development and the GDS development team, which is responsible for creating robust production implementations of those algorithms. The ASTER project is an international collaboration. The instrument is being built in Japan, under the auspices of the Ministry of International Trade and Industry (MITI), and will fly on a US satellite. ASTER data will be processed in both Japan and the United States. The Science Team is made up of members from both countries as well, with algorithm development input coming from scientists in Japan and diverse locations in the U.S. The GDS Team's main interaction with Japanese developers has been with regard to the Level-1 data product format, which has not yet been finalized.

2.2 EOS Environment

As part of the EOS System, the ASTER PGS is required to comply with EOS software standards. In an effort to avoid compatibility problems and excessive use of resources, the EOS environment imposes a host of restrictions on production software. The restrictions include the choice of programming language, the implementation of user supplied parameters and the format of archived data files. All EOS production software must use a set of software libraries and utilities known as the Science Data Processing (SDP) Toolkit. System services, production scheduling, and file access mechanisms are provided and regulated by the Toolkit. Some of the choices made while implementing the ASTER PGS design are due to the fact that the SDP Toolkit is being developed concurrently with production software for the early EOS platforms.

ASTER is unique among the AM-1 instruments with regards to data processing flow. First, Level-1 processing (i.e. generating geometrically and radiometrically calibrated radiance at "GDS01" from raw satellite data) will take place in Japan.[2] Level-1 data will be "shipped" to the Land Processes Distributed Active Archive Center (LP DAAC) in South Dakota. ASTER higher-level data products, including all of the Standard Data products discussed in this paper, will be produced on-demand at the DAAC. On-demand production adds complexity to the scheduling process, due to dependencies between the ASTER products. Figure 2 shows the inter-product dependencies. The arrows between boxes indicate inter-dependencies involving the use of data, metadata, or Quality Assurance (QA) information from other products. ASTER GDS software is required to be capable of producing "best" products, with suitible notification in the metadata, even in the absence of some input data sets. In many cases, a "missing arrow" (with respect to Figure 2) would simply be noted in metadata or QA data.

2.3 External Data Sets

APGS processing uses data from external (non-ASTER) sources, primarily in the atmospheric correction algorithms. Digital Elevation Model (DEM) data is supplied via specialized Toolkit routines. National Centers for Environmental Prediction (NCEP) and Data Assimilation Office (DAO) atmospheric data sets describing temperature, pressure, humidity etc. will be available via the Toolkit as HDF files. Data from other EOS instruments, mainly MISR and MODIS, will also be used when they are available. The formats and precise contents of most of the external data sets have yet to be finalized.

2.4 Science Algorithm Environment

Another key factor that influences the design and implementation of ASTER software is that the some of the science algorithms are still being optimized. Flexibility to adjust for algorithm changes and accommodate various forms of test data is therefore extremely important. There is a great deal of communication between the PGS developers and the scientists, much of which is via electronic mail.

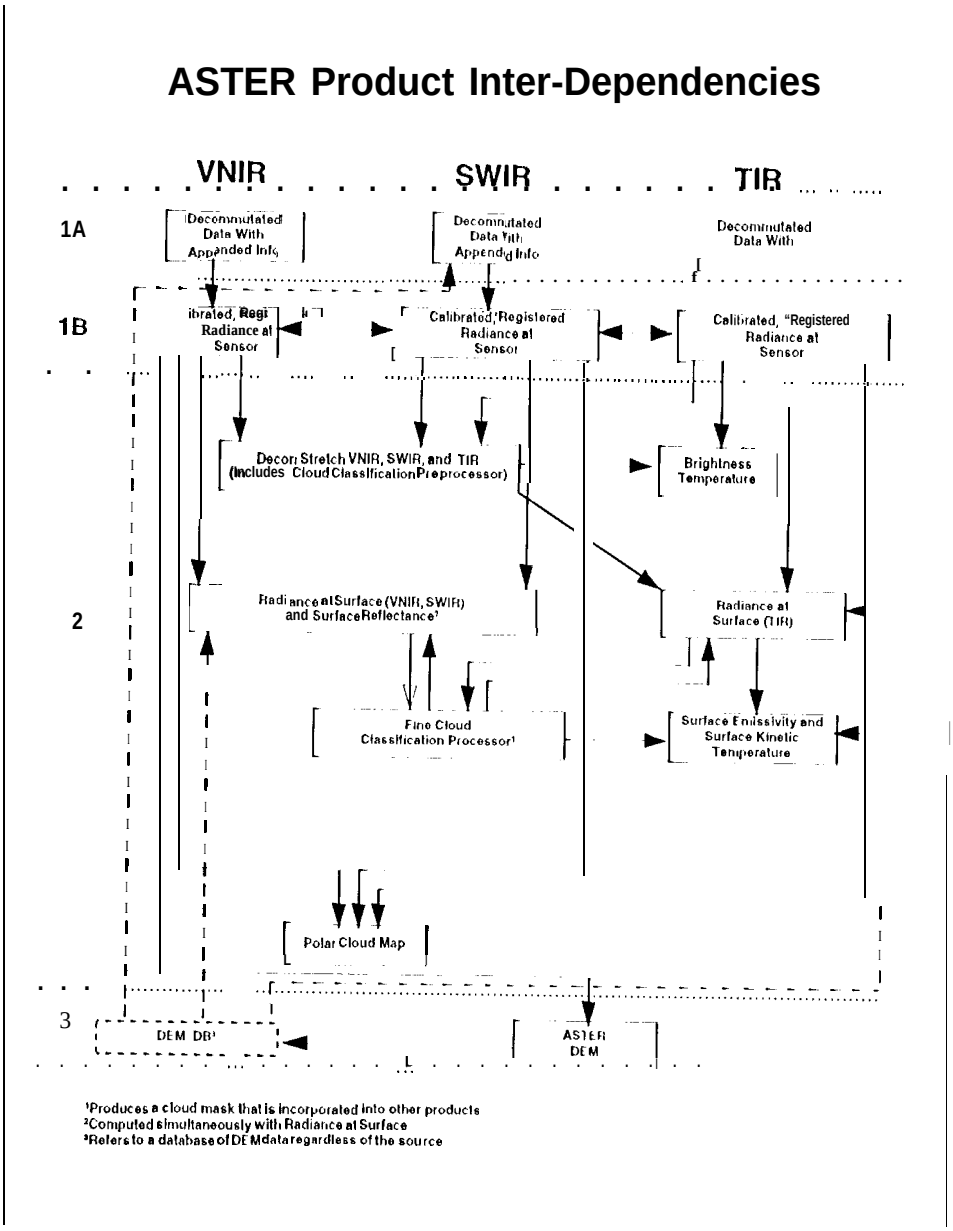


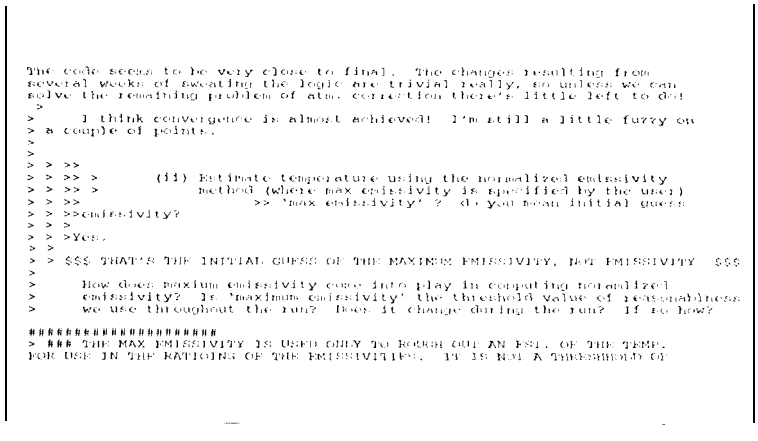
Figure 2: ASTER Data Product Inter-dependencies

3 METHODOLOGY

This section discusses the methodology that was adopted for the APGS including the developmental model, the configuration management approach, and the testing approach.

The ASTER methodological approach developed out of concerns relating to the ASTER, EOS, and Science Algorithm environmental issues that have been discussed previously. The APGS development schedule required that the production software should be designed and developed concurrently with the development of the Science Algorithms themselves as well as the supporting EOS Toolkit environment. Multiple developers were to work on interlocking and interacting low level modules that would be expected to change as new EOS and Science Algorithm requirements and prototypes were received. Many critical system elements were subject to modification during design and implementation, both high level algorithmic elements as well as low level elements such as the structure and format of data sets and metadata. In addition both the format and content of a number of external interfaces were unavailable during the APGS design process. A large, dispersed science and engineering community was interested and involved in the development and review of the Science algorithms and the APGS requirements in general. Finally, the APGS was required to be platform and system independent and the installation and acceptance testing of the operational system were planned to be carried out remotely.

The APGS design team adopted a Rapid Development Methodology[3] that would provide the flexibility to meet these demands. The system life cycle utilized a modified spiral development model rather than the classic waterfall. Multiple deliveries were scheduled with the most stable elements to be implemented first. A fundamental principle of the design was to plan for change and redesign. Incomplete, unstable, or unknown requirements and interfaces were identified and were isolated or encapsulated wherever possible. In order to obtain rapid consensus of requirements and design issues an (I)CI) dialogue or perpetual town hall system was introduced that connected most of the ASTER Project via e-mail. Requirements presented in draft form over the Internet allowed early clarification and extended dialogue as to functionality, performance and implementation issues. The use of this open dialogue was a critical element in the success of the concurrent engineering requirements of the Project; a frequent dialogue on requirements and design issues was conducted with the entire history of each interaction recorded and instantaneously available. During these discussions the Algorithm Developers, the science community, and designers and developers were involved and as a result of the rapid feedback the designs and implementations were highly consistent with the final intent of the requirements. An example of an actual e-mail exchange regarding a feature of the Emissivity algorithm is shown in Figure 3.



```
The code seems to be very close to final.  The changes resulting from
several weeks of sweating the logic are trivial really, so unless we can
solve the remaining problem of atm. correction there's little left to do!
>
>     I think convergence is almost achieved!  I'm still a little fuzzy on
> a couple of points.
>
>
> > >>
> > >> >       (ii) Estimate temperature using the normalized emissivity
> > >> >            method (where max emissivity is specified by the user)
> > >>                    >> 'max emissivity' ?  do you mean initial guess
> > >>emissivity?
> > >
> > >Yes.
> >
> > $$$ THAT'S THE INITIAL GUESS OF THE MAXIMUM EMISSIVITY, NOT EMISSIVITY  $$$
>
>     How does maxium emissivity come into play in computing normalized
>     emissivity?  Is 'maximum emissivity' the threshold value of reasonableness
>     we use throughout the run?  Does it change during the run?  If so how?

######################
> ### THE MAX EMISSIVITY IS USED ONLY TO BOUND OUT AN EST. OF THE TEMP.
FOR USE IN THE RATIOING OF THE EMISSIVITIES.  IT IS NOT A THRESHHOLD OF
```

Figure 3: Excerpt from design-by-email interaction

In addition the APGS designers were able to suggest changes and enhancements to the EOS system developers that proved to be of benefit not only to the ASTER project but to other instrument teams that were earlier in the development cycle.

The configuration management model required platform independence as well as multiple independent developer activity without risk of data aging for the chosen CM approach. The APGS team chose the freeware shapeTools that provided dependency ("*make*") specification with sharing of object files in independent directory trees as well as positive version and Build configuration control.

The Rapid Development Model establishes a cycle of testing that becomes increasingly formalized as deliveries progress. The APGS testing approach follows this general model, with testing of the initial (Beta) delivery being informal (as relates to requirements) and succeeding deliveries scheduled for increasingly formal requirements testing. The Beta delivery testing also focused on evaluating platform independence (Sun SPARC station vs. SGI Power Challenger), and establishing a remote testing protocol that could be used for future deliveries. These issues are discussed later in more detail.

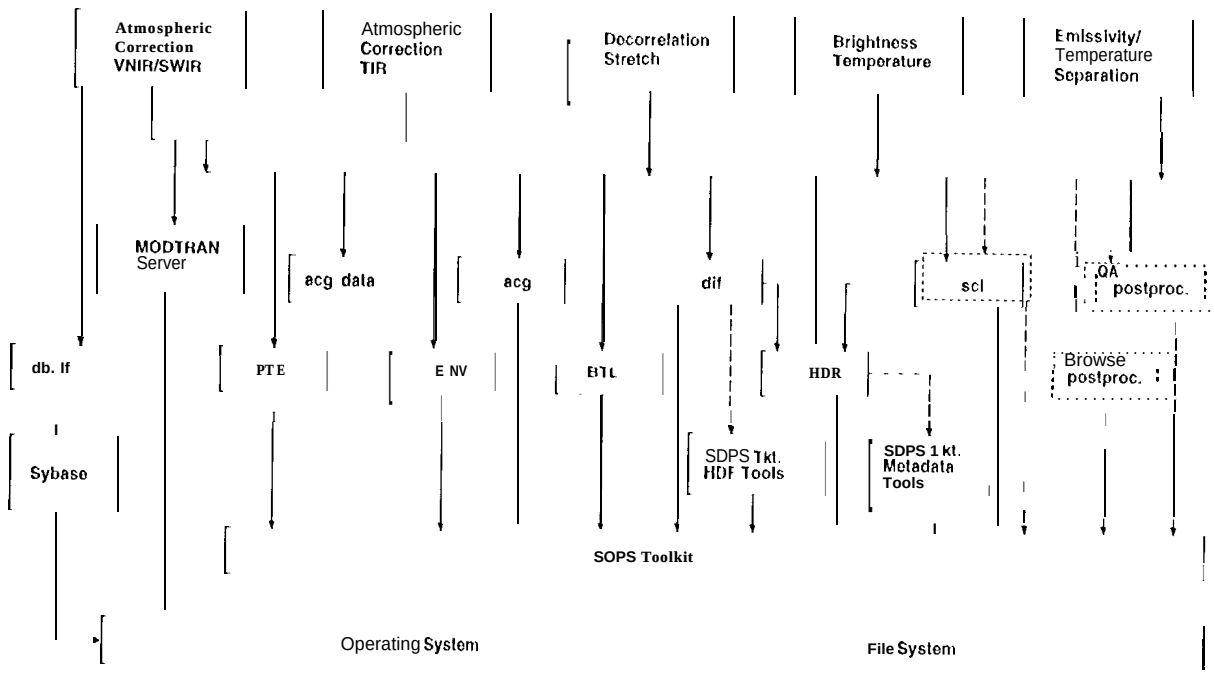
4 ARCHITECTURE

There are a few distinctive features in the ASTER PGS design which we feel have greatly contributed to the success of the development effort to date: Careful design of the overall architecture and repository directory structure up-front, strict usage of modular design and information layering, designing for platform independence, and designing for functional flexibility to cope with algorithm and tool changes.

The design process began with each scientist responsible for an ASTER data product supplying the PGS Team with either prototype code or an algorithm specification. An important design step was to look for functionality common to two or more programs and relegate those functions to libraries. This minimized code duplication and made the most efficient use of a small development team. Figure 4 shows the APGS hierarchy of Product Generation Executables (PGE) and libraries. The five large boxes on the top row are the PGEs. The PGE labeled Atmospheric Correction VNIR/SWIR produces surface reflectance as well as surface radiance, and the Emissivity/Temperature Separation PGE produces both Surface Emissivity and Surface Kinetic Temperature data products. Boxes in the lower tiers represent libraries called by the PGEs.

One extremely significant design decision was made for the VNIR/SWIR surface radiance and reflectance algorithm. This algorithm uses a Radiative Transfer Code (RTC) to model the effects of aerosol and molecular scattering in the atmosphere. It was determined that the processing requirements for running the RTC a large number of times per data product were prohibitive. A lookup table (LUT) approach, using pre-calculated RTC values, was chosen. However, in order to achieve sufficient accuracy, multi-Gigabyte tables are required. Sybase, a commercial relational database management system (RDBMS) was incorporated into the APGS design to manage the tables efficiently. In addition to simplifying LUT access by the APGS software, Sybase provides a reliable client-server architecture which can be used when populating the database. This gives us the capability to populate the LUT using many computers in parallel, with each machine interacting with the database via the Internet. The Sybase server keeps track of which runs have been completed and provides new input parameters to the RTC machines as those machines become available. In actual production, the LUT is not expected to require frequent updates. However, if there is a large volcanic eruption or other drastic change to the atmosphere, an update may be required. If the update is urgent, the process can be accelerated by simply running the RTC on more client machines. The LUT itself, and the benefits of using an RDBMS for LUT implementation, are described by Murray, Eng and Thome.[4]

The SDP Toolkit interface, with its somewhat complicated mechanisms for accessing Run Time Inputs (RTIs), messages, and files, could easily have led to code which was difficult to maintain. We were able to avoid these problems by carefully customizing our configuration management (CM) system to adapt to the Toolkit methods. We also tried to insulate ourselves from Toolkit changes by moving Toolkit calls to as low a level as possible in the design. If Toolkit functionality or interfaces change, we only need to change low level library routines, not the whole design.



ASTER Product Generation System, Software Architecture. Version 1 additions shown in dotted lines

Figure 4: APGS Library Modules

When a designer needed a functional module that was not defined due to incomplete specifications or not yet implemented Toolkit functionality, a "dummy" routine was created. Each of these S() called "stubs" were given input and output parameters based on the best available information. The tendency was towards simple, straightforward interfaces, in the hope that the actual interface would be very similar and require minimal code changes to higher level routines. This approach has worked well, resulting in very little code being thrown away.

The individual PGEs and libraries were designed using structured analysis and design methods[5] and Computer Aided Software Engineering (CASE) tools. Data flow diagrams and structure charts were generated for each program and reviewed internally.

5 IMPLEMENTATION

Experience gained from other collaborative software efforts has shown us the importance of working in an integrated structure from the very beginning. By taking the time to define a directory hierarchy suitable for automated builds of the entire system, we avoided frustrating periods of local integration. Each developer worked on his modules in place, using "virtual file system" features of the configuration management system. Having a consistent directory structure used by all PGEs allowed complex build procedures to be shared and maintained in a single place in the hierarchy. Every effort was made to avoid difficult-to-maintain code cloning.

The CASE tools provide a capability to generate program skeletons based on structure charts created during the detailed design phase. The code generation tools are customizable so that the resulting code can be created in the proper directories from the very beginning. Another valuable tool provided by the CASE system was a reverse-engineering tool, which produces structure charts from C code. We used the reverse engineering tool extensively during code and design iterations.

All compiled APGS software is coded in ANSI C with one exception. MODTRAN is a very large Fortran program (over 15,000 lines) which is used by the atmospheric correction algorithms. Waivers were obtained from EOS to use MODTRAN as-is with only minor modifications made to accommodate SDPS Toolkit file I/O. Keeping MODTRAN as intact as possible minimizes the effort required to incorporate updated MODTRAN versions during the life of the project. All other code complies with EOS coding standards completely.

Run Time Inputs (RTIs) are the mechanism by which processing options and data-set selections are implemented in the APGS. RTIs provide the flexibility required for scientific experimentation and validation, as well as the only means by which data requesters can specify non-standard processing options. Each PGE has a set of options which can be set at runtime via assignment statements in the Process Control File (PCF). These options can be based on policy set by the Science Team (such as the MODTRAN version to be used), or in some cases, user preferences. These preferences will be propagated from the user's data processing request (DPR). The use of RTIs in APGS code also allows science algorithm developers to experiment with the actual production software, without having to rebuild it in a "debug" mode, which may behave differently than production code.

PGS software is required to be able to run on an assortment of Unix-based computers. The APGS Team has addressed this requirement by developing all software to run on at least two different platforms (Sun/Solaris and Silicon Graphics/Irix).

5.1 Configuration Management

This subsection gives a description of the main features of our Configuration Management (CM) system, and delves into some detail about some of the unusual features of the system. The CM system provides efficient version control for collections of files making up higher-level entities. It also provides unusual features which facilitate productivity in a very heterogeneous development environment.

5.1.1 Version Control

We are using a free software system called shapeTools for configuration management (CM). Like RCS and other common Unix CM tools, shapeTools provides file-level revision control, histories, logging, locking, and retrieval based on version. These capabilities by themselves do not suffice in establishing a concept of a 'version' for a component or system consisting of more than one file. The shapeTools package provides aliasing and user-defined attribute capabilities which provide a basis for defining and controlling a multi-file entity. Also part of the package is the program *shape* itself, which performs essentially the same functions as the program *make*, but with knowledge of the files configured under the shapeTools system. Our CM system consists of a set of SDPS scripts and makefiles as well as the shapeTools programs.

The APGS software architecture consists of Product Generation Executables (PGEs), executable programs, and common software libraries. In addition to the software itself, there are several types of scripts and data files which are a necessary part of any delivery. A PGE may consist of more than one executable program, bundled together in execution by a shell script. In order to define a 'version' of a PGE, we have to be able to associate a version of every file that is required to build the PGE - including source files that make up common software libraries. In addition, a version of a PGE must include scripts and data files which the PGE requires to operate,

as well as test scripts and data. This association is accomplished using an alias, and assigning that alias to the appropriate version of every file that is part of, or required by, the PGE. Once a version of a PGE is so defined, that version may be retrieved at any time from the repository using a single command. Note that in this scheme a single version of a file may belong to more than one version of a PGE; this is frequently the case with files belonging to common software libraries. The shapeTools system allows a single version of a file to have several aliases. In addition to defining versions of PGEs, we define builds of the whole system using the same technique.

Normally deliveries of APGS are made as an entire build, but we do deliver patches to the system as needed. Patches are controlled in the system using a similar aliasing scheme to that used for PGEs and builds.

5.1.2 Productivity Tools

Our CM system was built with productivity in mind. It includes a script which allows a user to have an arbitrary shapeTools program executed throughout the entire repository tree. It includes *make* targets which allow developers to easily assign aliases to all files required to build a PGE, or to generate source code counts, calling-trees or ANSI-C prototypes.

Our CM system also includes a system of makefiles which include logic to do any of the various types of installations or builds needed in the system. This allows a developer to create a new makefile in minutes by simply modifying a template makefile which includes the system makefiles.

5.1.3 Integration Capabilities

The APGS consists of many disparate elements: C and Fortran source files, shell scripts, perl scripts, SQL scripts, parameter files, and data files. There are pieces of information which are shared among many of these types of objects, and this can have an impact on system maintainability. For example, in the environment established by the SDP Toolkit, a PGE can refer to a file only by logical file number. At execution time, the Toolkit uses a mapping between logical file numbers and physical file names to resolve references the software makes to logical files. The mapping is contained in a parameter file called a Process Control File (PCF). There may be many PCF files for a PGE, each mapping a given logical number to a physical file. If for some reason the logical number has to be changed in the software, the many PCF files have to be edited in order to update the logical number. We have solved this problem by processing PCFs and scripts of all types with the C preprocessor. This allows us to reference C named constants in PCFs, shell scripts, SQL scripts, perl scripts, or IDL scripts. If a logical file number changes, it is changed in one place - a C header file. Then a remake of all of the scripts and data files causes the change to be distributed wherever it needs to be. This processing is implemented using *make* and the system makefiles.

A side effect of this technique has been to allow the use of structured, modular and information layering techniques in the development of complex shell scripts and SQL scripts. Shell 'subroutines' are stored in C header files as C macros, and then used repeatedly in other script files. This has greatly aided the maintainability of our system installation script, discussed below, which uses many shell functions, and many constants from C header files throughout the system. We also have a shell test script template, implemented as a C macro, which allows developers to implement sophisticated test scripts very quickly by defining shell functions and then including the template.

Using the C preprocessor on SQL scripts has greatly reduced the development time and enhanced the maintainability of the project SQL code. There are many instances where pieces of information are shared between SQL code and C code, the meaning of a status field in a database table, for example. Again, modularity of coding is greatly facilitated by defining often-used chunks of SQL code as a C macro and then including the header file

containing the definition in various SQL scripts. In a typical example, the configured version of the SQ1, program which performs a database query is 250 lines of SQL code and comments. The total number of lines of code and comments in all header files included in the SQL script is 953. The actual SQL script generated by running the C pre-processor on the configured version contains 1989 lines of code and comments. If code generation rates based on line count are reliable productivity indicators, the pre processor technique yields a productivity increase in excess of 100% in this case.

6 INTEGRATION AND TEST

6.1 Installation Support

In order to facilitate the installation process, we developed a script which performs all of the work of installation of the APGS. If all of the necessary support software is installed on the system, the APGS (not including the Sybase database) can be installed with a single command. The installation script first determines which type of machine and operating system it is running under, and sets a variety of compiler and linker flags based on that determination. These flags are all stored in environment variables. It then looks for support software: ANSI C and Fortran compilers, GNU *make*, and libraries. After configuring the necessary environment variables, the install program runs *make* from the top of the directory structure. This builds the entire system. We have installed the system successfully on Sun workstations, SGI machines running IRIX6.x, and on a DEC Alpha. It is interesting to note that, while our makefiles work under both the Sun and GNU versions of the *make* program, they do not work under the SGI or DEC Alpha versions of *make*. Fortunately GNU *make* runs on all of the platforms. We have therefore included GNU *make* as part of our delivery package.

6.2 Test Automation

Each of our PGEs has a suite of tests meant to demonstrate basic functionality. The suite of tests is run whenever an installation is made, and during development for regression testing purposes. Since they are run many times, we decided to take the time to automate testing. For each of the PGEs, we have a shell script which allows a user to run the entire suite of tests or run a selected subset of tests, and run the tests normally or under a variety of debugging conditions. The test scripts are self-documenting; they can be made to only print information about each test but not actually run it - this provides a sort of simple "oil-lill(> test plan" capability. The test scripts also perform an automated comparison of outputs to expected results files. For many cases the entire data product files are simply compared bit for bit. In other cases, the test scripts compare only the header information in the files, and then generate IDL scripts which compare the bodies of the data product files with those of the expected results files. This feature provides a way to quickly find out if the suite of tests ran as expected, in a way that is not graphics-intensive and so is amenable to the remote testing environment. The test scripts are all based on a set of header files containing shell program functions and commands, parameterized with C pre-processor macros. To generate the test script for a particular PGE, the developer must generate the bodies of two shell functions for each test: one function which documents the test by printing the objectives and expectations of the test to the screen, and the other function which performs any needed post-test processing, such as comparing output files with expected results files, generating IDL scripts, or printing excerpts from the log files. Having defined these functions, the developer includes a header file containing the template for the test script, and the C pre-processor does the rest.

6.3 APGS Beta Integration and Test

The first major delivery of the entire suite of APGS software was made in February and March of 1996. As mentioned earlier, ASTER data will be processed at the LPDAAC at EROS Data Center (EDC) in South Dakota. EDC is "fill" from Pasadena, California where the software is being developed. In order to save time and travel expenses, our goal was to complete the Beta Integration and Test entirely via remote access.

A great deal of preparation is necessary for a successful remote integration. We began by establishing a good working relationship with the key people at EDC and Hughes (the Toolkit and LPDAAC-hardware contractor). They helped us get a thorough understanding of the target hardware and were responsible for setting up the necessary user accounts as well as installing supporting software.

Before delivering to EDC we made simulated deliveries to test accounts on our machines at JPL. Many potential problems were fixed in this "local delivery" stage. We were able to build and run most of the software on both SUN and SGI platforms. The only PGE we couldn't test on a local SGI machine turned out to be the one that caused us the most trouble during the actual integration. We did not have Sybase client software for our SGI machine and could not run the VNIR/SWIR atmospheric correction algorithm for that architecture locally.

The multi-Gigabyte LUT mentioned earlier was the only part of the APGS that was too large for Internet file transfer to be practical. We made a tape "snapshot" of the Sybase database containing the LUT, four days before the integration was to begin. The tape was shipped to EDC and installed by EDC personnel using scripts provided by the APGS Team.

On February 29th we began installing the APGS Beta software. By the end of the day we had built the entire system at EDC using the Science Computing Facility (SCF) version of the SDP Toolkit. During the next two weeks we ran tests and fixed bugs by uploading replacement files and rebuilding. The SCF Toolkit integration was completed on March 13th.

EDC and Hughes personnel continued testing and benchmarking the software for another month; including building with the Product Generation System (PGS) Toolkit, a prototype of the future production Toolkit. The process was wrapped up with a delivery-complete tele-conference between JPL, NASA-Headquarters, EDC and Hughes on April 12th.

7. CONCLUSION

The ASTER PGS development team has demonstrated a successful beta implementation of the ASTER standard data product production system using a rapid development methodology. In this paper we have described the requirements environment in which the software was designed, the methodology, and key features of the architecture and implementation. A few features worth highlighting are: the use of the C preprocessor and a CM system to maintain non-C files (shell scripts, SQL, code and static data files), and the use of the flexible client-server capabilities of a relational database to implement a highly parallel approach to populating the atmospheric correction lookup table. We also described our Integration and Test approach using remote access via the internet.

In summary, we feel that the rapid development model was very suitable for this project. Future software deliveries (Versions 1 and 2) will require more testing than the beta procedures described in this paper, but we expect that the flexibility designed into the system will enable us to integrate future releases as successfully as the beta version.

8 ACKNOWLEDG MENTS

The research described in this paper was carriedout by the Jet 1 'ropulsion 1 aboratory, California Institute Of Technology, under a contract with the National Aeronautics and Space Administration. The Al 'GS Team would also like to acknowledge the contributions made by the late Charles Voge as the lead member of the APG S Team. I lis guidance was invaluable.

9 REFERENCES

[1] A.B. Kahle, et. al. *The Advanced Spaceborne Thermal Emission and Reflectance Radiometer (ASTER).* International Journal of Imaging and Technology, Vol3, 144-156, 1991.

[2] H. Tsu, Y. Yamaguchi, A.B. Kahle *ASTER Science Mission Overview.* SPIE Conference, Denver, August 1996.

[3] Riley, B. Norman, and W.H.Spuck. *System Engineering and the Rapid Development Method* . Proceedings, Fourth Annual International Symposium, National Council on Systems Engineering (NCOSE), 1994 .

[4] A. Murray, B.T. Eng, K. Thome, *Implementation of a Very Large Atmospheric Correction Lookup Table for ASTER Using a Relational Database Management System* . SPIE Conference, Denver, August 1996 .

[5] M. Page-Jones, *The Practical Guide to Structured Systems Design* . 2nd Ed., (Prentice-Hall, Englewood Cliffs, N.J., 1988) .